COMP 307 Assignment 2: Neural and Evolutionary Learning

Part 1: Neural Network

1. **Report the output and predicted class of the first instance in the dataset using the provided weights.**

```
Epoch:  1
Raw Values of first forward pass:  [0.47627171854273365, 0.5212308085921598, 0.47617240962812674]
Predicted Class: Gentoo
```

My network incorrectly classified the first instance, although it rated the chances of it being all three different classes very similarly, as all three values are reasonably similar.

2. **Report the updated weights of the network after applying a single backpropagation based on only the first instance in the dataset.**

```
Weights after performing BP for first instance only:
Hidden layer weights:
 [[-0.28052064 -0.21971835]
 [ 0.07839682  0.20086728]
 [-0.30115025  0.32062226]
 [ 0.09937879  0.01033606]]
Output layer weights:
 [[-0.27752534  0.01757923  0.19865828]
 [ 0.0941994   0.11586195 -0.37290981]]
Accuracy: 51.49%
```

3. **Report the final weights and accuracy on the test set after 100 epochs. Analyse the test accuracy and discuss your thoughts.**

```
Epoch:  100
Accuracy: 82.84%
After training:
Final Hidden layer weights:
 [[ 0.93300053 -9.81238895]
 [-7.29027973  5.20341616]
 [ 2.38938873 -1.40616717]
 [ 2.47148091  1.43004753]]
Final Output layer weights:
 [[ -9.67263879  -2.44486417   3.24212769]
 [  4.90797584  -2.87370739 -11.64832746]]

Test Set accuracy: 81.54%
```

After 100 epochs, the neural network had not converged and reached a final training accuracy of 82.84%. The training accuracy increased less per epoch the more epochs that occurred, increasing by 7.84% between the third and fourth epochs, and only increasing by 0.38% between the 96th and 97th epochs.

This suggests that the weights were changing very little towards the end. The test accuracy was 81.54% which is very good in comparison to the training accuracy. The fact that both training and test accuracies were within 2% of each other is a strong sign that the network is not overfitted and has generalised well, which is indicative of a good neural network.

4. **Discuss how your network performed compared to what you expected. Did it converge quickly? Do you think it is overfitted?**

   My network performed about how I expected, getting a good accuracy on both the test and training sets.

   My network did not converge within the 100 epochs and given the decreasing difference between each successive run I don't think it would within any reasonable number of epochs. I believe that if I randomly generated the initial weights, I could get lucky and create a better starting set of weights which could hypothetically converge quicker, possibly within the 100 epochs.

   I do not believe my network is overfitted. I talked about it briefly in the answer to question 3, but the main reason I do not think it is overfitted is that the test and training classification accuracies are very similar to each other. In an overfitted network, the classification accuracy on the training set is often a lot higher than the classification accuracy on the test set as the network has specialised to the initial training instances by including outliers in this data, and so an overfitted network generally does worse on the test set by a substantial margin, this margin increasing the more overfitted the network is, and I see no evidence of any of this in my network or in its predictions and results. Hence, I do not believe that my network is overfitted.

5. **Add bias nodes into the above network (with the below initial weights). Train it using the same parameters as before, and report your test accuracy. Compare the accuracy achieved to that of the original network and discuss possible reasons for any performance differences.**

```
Epoch:  100
Accuracy: 99.25%
After training:
Final Hidden layer weights:
 [[  0.73891152 -13.79095233]
 [ -9.19373558   6.7716038 ]
 [  4.04264096  -0.28302753]
 [  4.4119594    2.85373954]]
Final Output layer weights:
 [[ -9.25294729  -7.26584961   8.19693274]
 [  9.24085017  -8.63857542 -10.96591619]]

Test Set accuracy: 100.00%
```

The training classification accuracy of my network with biases was 99.25%, and the test classification accuracy was 100%. This is far superior to that of the network that did not include biases, which scored 82.84 and 81.54% classification accuracies on the training and test set respectively. This is a big of an improvement as is possible, as it is not possible to achieve more than 100% accuracy, so the network with biases was far superior to the network without them. One of the main reasons for this is because adding biases means that if an input consists of zeros, without a bias the network cannot influence the final prediction, as without the bias every result will be zero, as zero times anything is zero. Adding biases solves the problems with this kind of input. However, the main reason that

biases are helpful is that they allow us to shift the entire activation function to the left or right, that is decrease it or increase it to better fit with the data. It essentially functions as the c, or constant, in the equation y = mx + c, which is necessary. Take for example if we are always getting results of -1, 0, and 1, when we are classifying based on the values 0, 1, and 2, we can just increase the weight of the bias until all the results shift up by one, therefore allowing us to classify everything more accurately, hence the improvement in classification accuracies across both the test and training sets.

Part 2: Genetic Programming for Symbolic Regression

NOTE: I have used code from
https://github.com/DEAP/deap/blob/master/examples/gp/symbreg_numpy.py
and https://deap.readthedocs.io/en/master/tutorials/advanced/gp.html
In building my genetic programming code in this section

1. **Justify the terminal set you used for this task.**
   I used the x value, and random numbers generated between 0 and 4. I chose this range of numbers as during exploratory data analysis I found the correct equation for the set of points provided to us, $x^4 - 2x^3 + x^2 - 1$, and since I have a negative function in the function set, I do not need any negative numbers in the terminal set. Hence, I have excluded them.

   The reason that I have included constants is to allow for the cases where the equation we are searching for has a constant term at the end, as without these constants my program may generate an equation with the same curve but offset from the target equation.

2. **Justify the function set you used for this task.**

   The initial function set I used consisted of add, subtract, multiply, protected, division, and negatives. I choose to include the four basic arithmetic operators in the function set as a starting point, although I included a special type of division. The reason I included protected division is that protected division is a special type of division that stops you from dividing by 0, which is helpful as it allows the tree to have closure. What this means is that any output generated by any function in the function set should be a valid input to every function in the function set, and normal division fails this, as if subtraction outputs a zero, normal division cannot accept this as the denominator as we cannot divide by 0. By using protected division I provided closure to the function set.

I also experimented with using exponents, cubes, and squares, but ran into difficulty dealing with a large number of invalid inputs to these functions (like (-2)^0.5, which creates a complex number) so decided to just exclude them as there were too many edge cases to deal with, as well as the fact that multiplication can generate exponents by itself.

However, when it came to running the program, I found that including protected division caused the algorithm to generate very deep trees that were often coupled with relatively high RMSE. Hence, since I know from my exploratory data analysis that the desired function did not use division, I chose to exclude it from my final function set. I am aware that this is mainly possible since the problem is univariate, which makes the EDA much simpler. If this was a multivariate function the correct equation may be much harder to find through EDA, so in that case, I would include protected division in the function set.

Therefore, my final function set was simply “add”, “subtract”, “multiply”, and “negative”.

3. **Formulate the fitness function and describe it using plain language (and mathematical formula, or other formats you think appropriate, e.g. good pseudo-code).**

The fitness function that I have used is Root Mean Squared Error or RMSE for short. I have used this as by squaring the error term, we punish the algorithms that are further away from each y value more harshly the further away they get, reducing the number of algorithms that are extremely incorrect on some points.

This function takes the difference between the predicted and actual value for each instance, squares it, adds all these values up then divides by the number of values to get the mean squared error, then square roots this value to give the final root mean squared error.

4. **Justify the relevant parameter values and the stopping criteria you used.**

The relevant parameters that I used are as follows:

Starting Population: 1000. This was chosen to ensure that I had a good chance of generating some good initial algorithms without being too large as to make the algorithm take too long.

Selection method: Tournament Selection. This was chosen to allow for a chance of low fitness algorithms to make it through to the next generation whilst making sure the best of the random selection also got through.

Generations: 100. This is my stopping criteria, as my program always runs for 100 generations, as this does not take too long in practice, but is long enough to generate algorithms with low RMSE. I empirically validated this by testing different generation lengths and found this to be the best balance overall.

Crossover chance: 50%, Mutation chance: 10%. I decided to go with crossover as my main method of evolving the algorithms, and have a small chance of mutation, as this represents a more natural view of the problem. I chose one point crossover specifically for this example too.

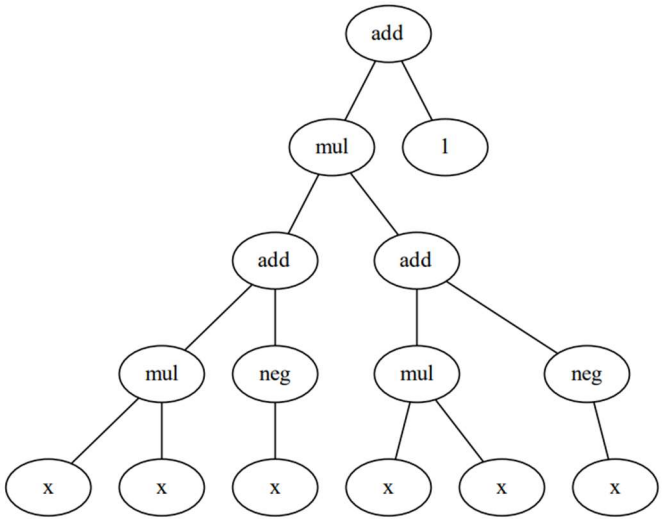
Max Depth: 10. I originally had this at 100, but I found that the python parser ran into a stack overflow if the depth was too high which was around 90, so I lowered the max depth to 10 as this seems more reasonable. Any tree deeper than 10 seems to me to be unnecessarily complicated.

5. **Different GP runs will produce a different best solution. List three different best solutions evolved by GP and their fitness values (you will need to run your GP system several times with different random seeds).**

I scheduled my program to perform 5 different runs, and below are the best three algorithms I generated, their fitness values, their tree, and the simplified equation. I have added code to my program that generates a pdf for each tree and displays the primitive tree in a graphical format to make it clearer what the final equation says, as the output of DEAP is hard to read for a human. Note that for the first two algorithms, they come up with the actual equation I discovered during EDA, however, due to what I believe amounts to a rounding error they do not have 0 root mean squared error.

First Equal:

Tree:



DEAP Equation: add(mul(add(mul(x, x), neg(x)), add(mul(x, x), neg(x))), 1)

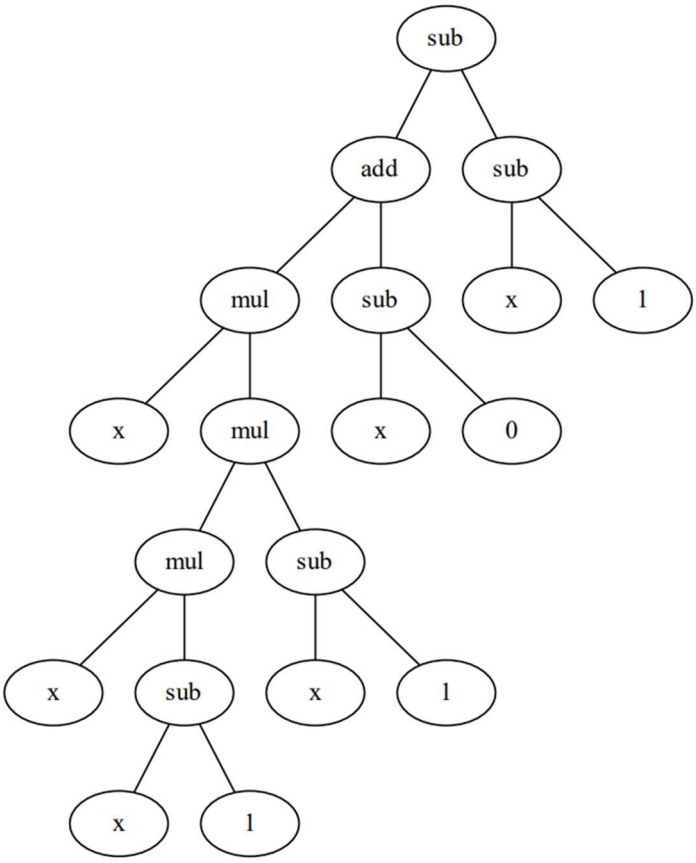
Math Equation: $\left(\left((x * x) + (-x)\right) * \left((x * x) + (-x)\right)\right) + 1$

Expanded form: $x^4 - 2x^3 + x^2 + 1$

RMSE: $2.65 * 10^{-6}$

First Equal:

Tree:



DEAP Equation: sub(add(mul(x, mul(mul(x, sub(x, 1)), sub(x, 1))), sub(x, 0)), sub(x, 1))

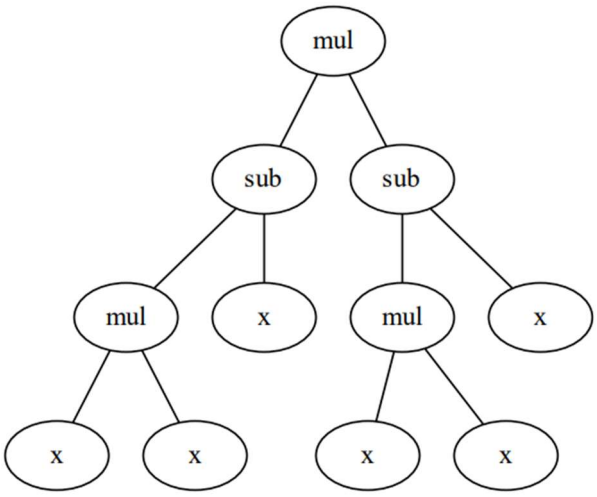
Math Equation: $\Big(\big(x * \big(x * (x-1)\big)(x-1)\big)(x-0)\Big) - (x-1)$

Expanded form: $x^4 - 2x^3 + x^2 + 1$

RMSE: $2.65 * 10^{-6}$

Third Place:

Tree:



DEAP Equation: mul(sub(mul(x, x), x), sub(mul(x, x), x))

Math Equation: $\big((x * x) - x\big) * ((x * x) - x)$

Expanded form: $x^4 - 2x^3 + x^2$

RMSE: 1.000